SOFTWARE COMMUNICATIONS ARCHITECTURE SPECIFICATION

USERS GUIDE

30 November 2010

Version: Next <Draft>

Prepared by:

JTRS Standards
Joint Program Executive Office (JPEO) Joint Tactical Radio System (JTRS)
33000 Nixie Way
San Diego, CA 92147-5110

REVISION SUMMARY

Version	Revision	Date
Next <Draft>	Initial Draft Release	30 November 2010

TABLE OF CONTENTS

TABLE OF FIGURES

1 SCA USER'S GUIDE

1.1 SCOPE

This Users Guide is intended to provide practical guidance and suggestions for developing SCA-compliant products. It is not a substitute for the SCA specification, but a companion document to provide guidance and rationale outside the structure of a formal specification. This document will expand in content and detail as user experiences accumulate with SCA Next.

2 SCA NEXT INTRODUCTION

2.1 SEPARATION OF WAVEFORM AND OPERATING ENVIRONMENT

A fundamental feature of the Software Communications Architecture (SCA) is the separation of waveforms from the radio's operating environment. Portability of waveforms is enabled by establishing a standardized host environment for waveforms, regardless of other radio characteristics. An example diagram of an SCA-based radio is illustrated in Figure 1. The waveform software is isolated from specific radio hardware or implementations by standardized APIs.

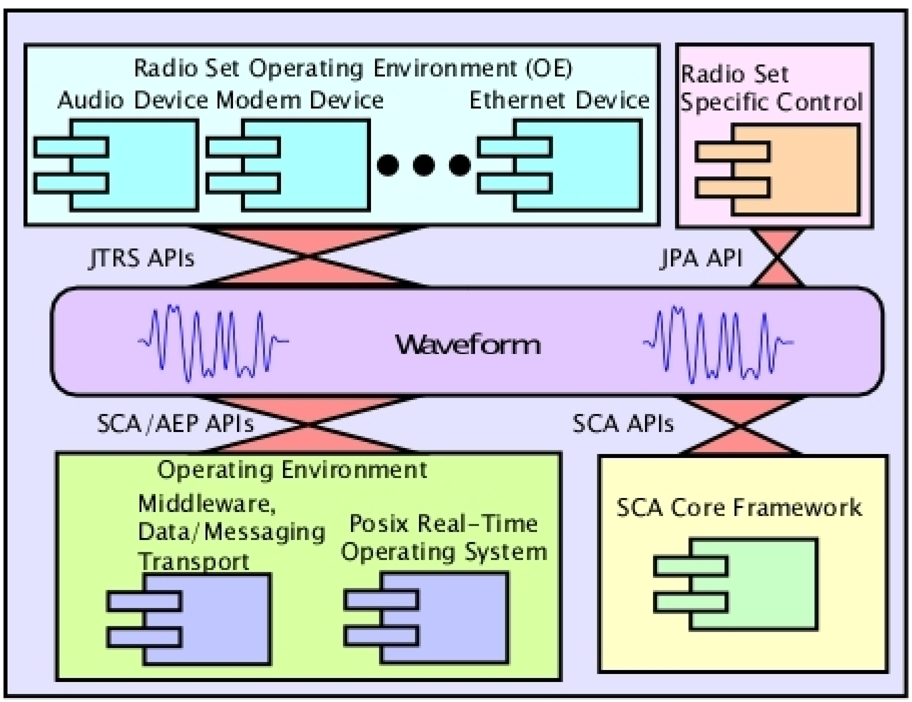

Figure 1: Example Radio Powered by SCA Next

2.2 OPERATING ENVIRONMENT

2.2.1 Application Environment Profile

To promote waveform portability among the many different choices of operating systems, the SCA specifies the operating system functionality in POSIX units of functionality. Specific operations such as *pthread_create*, *open*, etc., that waveforms/applications expect to be supported by the radio platform are defined as subsets of the IEEE POSIX specification. A radio developer is permitted to provide additional operating system functions, but the waveforms can only access the functions defined in the SCA's Application Environment Profile (AEP). This assures any SCA-compliant radio can execute the waveform.

SCA Next defines AEP and Lightweight (lwAEP) profiles, to support SCA conformance across a range of radio sets ranging from a small handheld to a multichannel radio embedded within an aircraft. The lwAEP is intended for very constrained processors such as DSPs that typically are not supported with more capable real-time operating systems. Some waveforms may require networking functions such as *socket* or *bind*. If the radio set is going to host networking waveforms, it must support the Networking Functionality AEP as an extension to the primary AEP profile (See [7] for additional information on networking).

2.2.2 Middleware and Data Transfer

In Figure 1, the radio platform provides middleware and data/messaging transport in addition to the real-time operating system. Middleware is a generalized service which facilitates messaging between software components, possibly hosted on separate processors. SCA 2.2.2 and its predecessors specified CORBA for the middleware and deferred the specific transport mechanism to the radio set developer. Historical data transfer mechanisms have been TCP-IP and shared memory. The former can introduce substantial latency and perhaps has unfairly tarnished CORBA's reputation within the radio community. A faster transport such as shared memory generally yields latencies acceptable for high-data rate waveforms. SCA Next has deleted the CORBA requirement and defined middleware-independent APIs, although they are still specified through interface definition language (IDL). Radio developers may use CORBA, or select a different middleware such as the lightweight Remote Procedure Call (RPC) used by the Android platform. Waveforms would require recompilation for different middleware implementations, but the APIs remain the same, sustaining waveform portability.

2.3 JTRS APIS

In Figure 1, several independent API sets separate the waveform from the radio set. The primary emphasis in API standardization has been upon the waveform-to-set interfaces illustrated in Figure 2. Only the interfaces between the waveform and the radio are standardized. The internal interfaces and transport mechanisms of the radio are defined as necessary by the radio provider. The intent is to provide portability or reuse of the waveform between radio platforms and not necessarily portability of the radio operating environment software. For additional discussion on waveform portability, see [5-6].

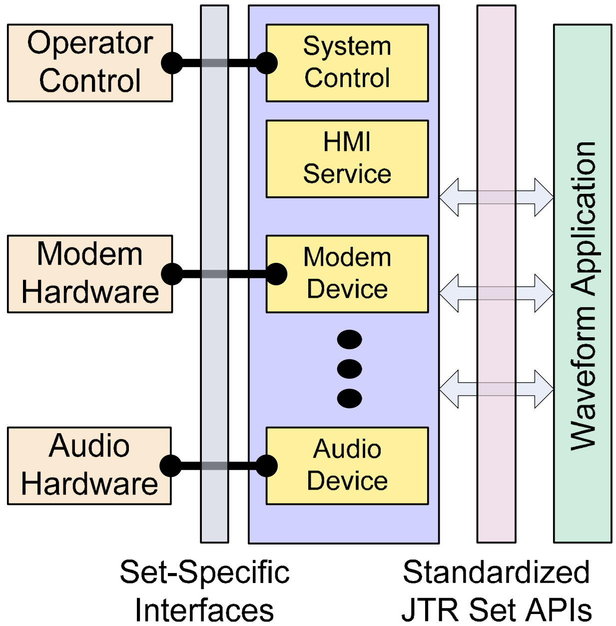

Figure 2: JTR Set and Waveform Interfaces

The SCA specification does not document the JTRS APIs which define services provided by the radio set to the waveform. These include services such as GPS, time, etc. A partial list of the JTRS APIs is provided in Table 1. The APIs have been developed with software design patterns to define a scalable and extensible infrastructure. See [2-3] for an introduction to the aggregation, least privilege, extension, explicit enumeration, and deprecation design patterns for JTRS APIs.

Table 1 Partial List of JTRS APIs

Audio Port Device API	Ethernet Device API
Frequency Reference Device API	GPS Device API
Modem Hardware Abstraction Layer (MHAL) API	Serial Port Device API
Timing Service API	Vocoder Service API
MHAL On Chip Bus (MOCB) API	Packet API
JTRS Platform Adapter (JPA) API	

The JTRS Platform Adapter (JPA) shown in Figure 1 is both an API and a design pattern for controlling the waveform by the radio set. (It is a particularly vexing problem, to define a portable command/control interface for waveforms across multiple radio sets.) This API uses the SCA property set as a container for waveform parameters controlled and manipulated by the radio set. It also supports bidirectional communication, permitting the waveform to provide status to the radio set.

2.4 SCA CORE FRAMEWORK

In all but the most constrained implementations, the radio platform provides a Core Framework with the general services depicted in Figure 3. Except for the Domain Manager, the services and implementations may be distributed among the various processors within the radio. The domain manager, being a singleton, maintains registries for the various hardware and software resources within the radio. It also provides interfaces for control by the radio control and management system, not defined within the SCA.

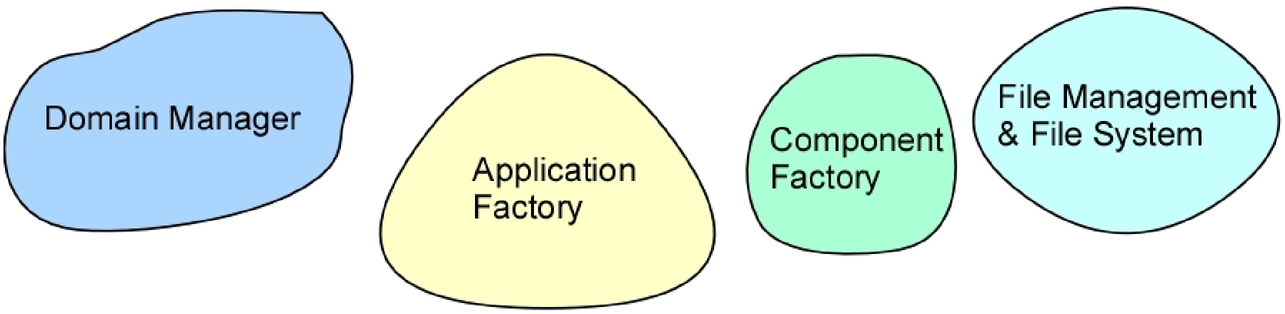

Figure 3: Core Framework Services

The component factory is an optional service that can be used to dynamically construct software services and components. It replaces the former Resource Factory with a more general constructor useful for applications, services, devices, and core framework elements.

The SCA defines a file system with global access – any software component anywhere within the radio can access a file located anywhere within the radio. SCA-compliant components are prohibited from using the native operating system functions to access files. This requirement increases the portability of waveforms and components because they can be relocated anywhere within the radio without having to modify file access. (Radio services and core framework services can access the native operating system file functions, but not waveforms.)

The application factory is typically the most complex service within the core framework. It is responsible for loading, instantiating, and connecting together the multiple software components of a waveform or application. Before describing the activities of the application factory in greater detail, it is necessary to first explain the concept of an SCA Device.

2.4.1 SCA Devices

An SCA device is a software interface for a piece of hardware which could be an Ethernet adapter, a modem adapter, or even the microprocessor itself. They are software proxies for hardware and intended to be used by multiple waveforms or applications. Every processor in the radio will have a Device proxy, as illustrated in Figure 4. The Device API promotes portability because the same *load* operation is issued to a Pentium, PowerPC, or OMAP processor.

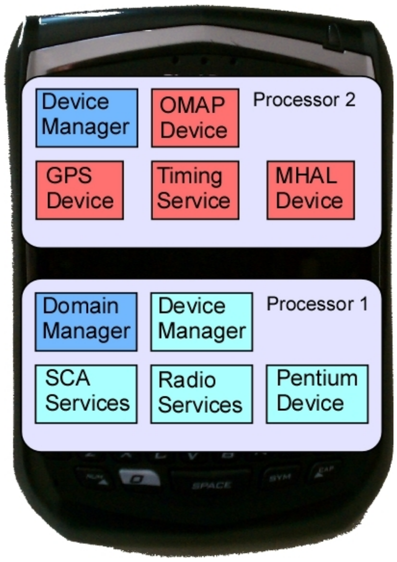

Figure 4: Devices within Example Radio

Also depicted in Figure 4 is the Modem Hardware Abstraction Layer (MHAL) device, one of the JTRS APIs. (The SCA does not mandate the MHAL – other modem devices and APIs can be used instead.) A Device Manager is a container for all of the Devices located on a processor. It also has responsibility for managing and reporting the status of the devices registered to it. In Figure 4 there are Device proxies for the Pentium and OMAP processors. The SCA defines the Device interface to facilitate loading new software components and binaries upon the processor.

In SCA Next, there are four types of device interfaces: Device, LoadableDevice, ExecutableDevice, and Aggregate Device. As the nomenclature suggests, they have escalating capabilities and features. In a departure from SCA 2.2.2, these different interfaces no longer inherit each other. Instead, the finer granularity of interfaces within SCA Next and the design pattern of optional inheritance discussed in 2.4.5 result in more lightweight components. Figure 5 illustrates the ExecutableDevice interface is an aggregation and no longer inherits from LoadableDevice as in SCA 2.2.2.

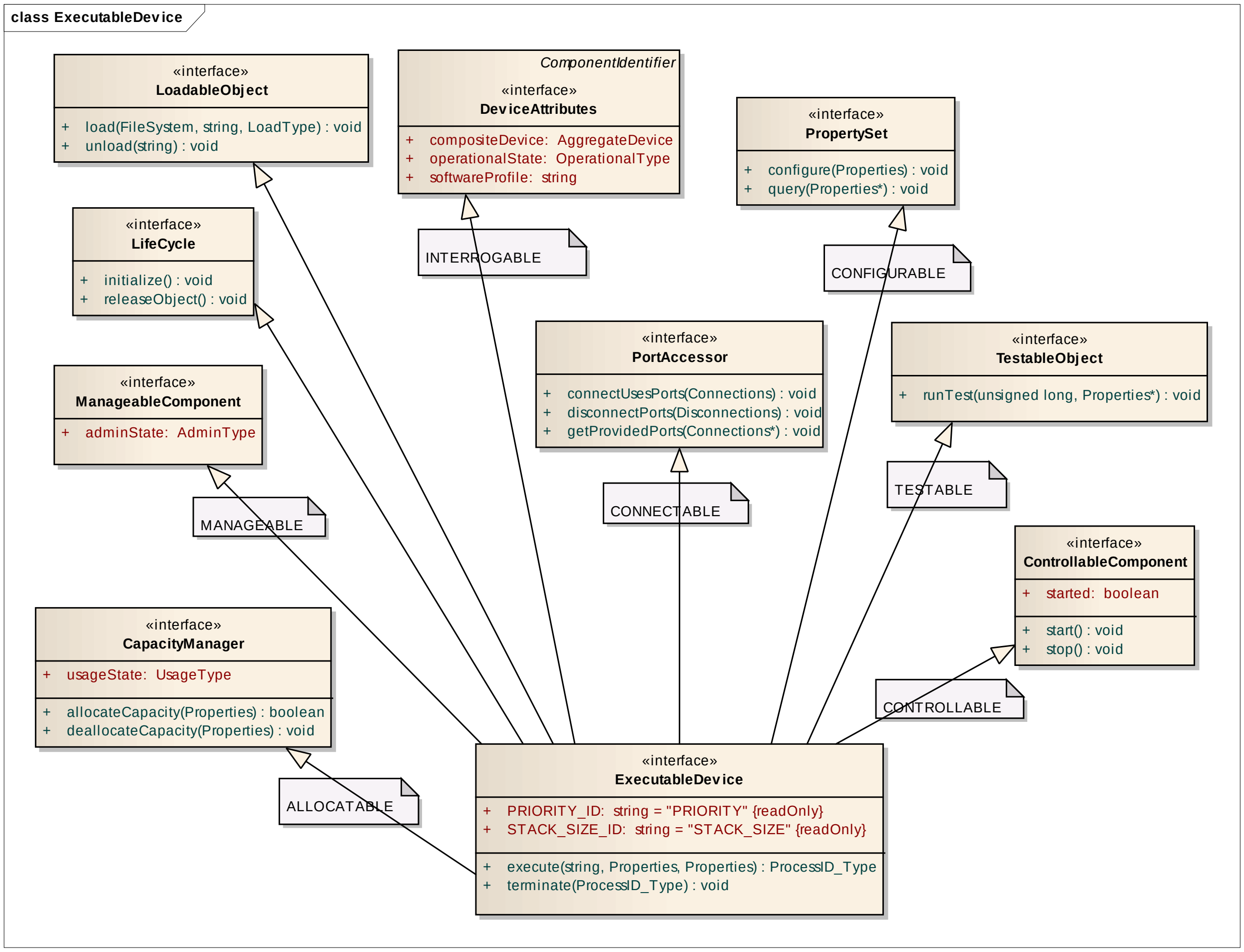

Figure 5: ExecutableDevice Interface UML

2.4.2 Domain Manager

The radio set's domain manager is responsible for controlling and managing all the hardware and software assets defined for the radio. The full SCA Profile is a plug-and-play profile that supports the registry interfaces for registering and unregistering elements to Domain and Device Manager components. This profile is suited for radio platforms where the hardware modules are plug-and-play and supports dynamic configuration. There are two additional profiles with decreasing levels of functionality as illustrated in Figure 6. The concept is that an SCA radio can be an almost-infinitely flexible platform with the Full Profile, or very minimalist with the Lightweight profile. In the Lightweight profile, the radio boots and begins executing a single waveform with minimal configuration and processing.

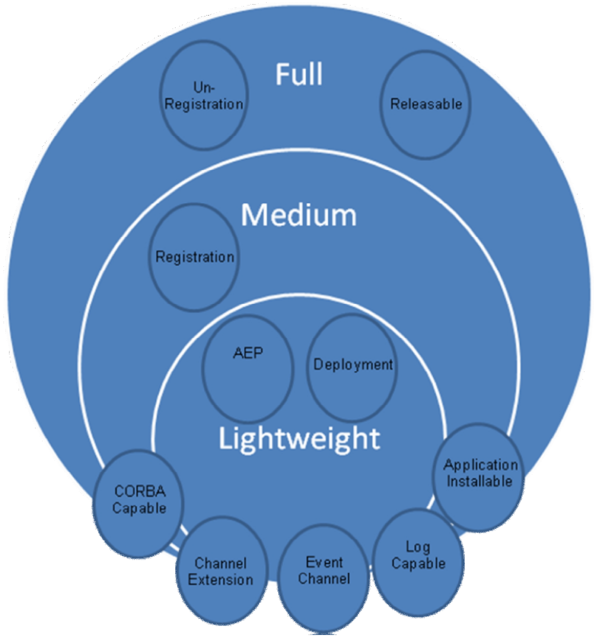

Figure 6: SCA Profiles with OE Units of Functionality

Additional units of functionality such as CORBA capable, event channels, log capable, etc., extend the capabilities of the radio set. These are independently specified in addition to the Full, Medium, or SCA Profile. The granularity of Figure 6 is a substantial change from earlier versions of the SCA, which had a one-size-fits-all model.

2.4.3 SCA Next Registration Model

Aside from deleting the CORBA requirement, the largest change in the new SCA version is the registration model. In SCA 2.2.2, the Domain Manager would query device managers and software components, embracing a ‘pull’ model. In SCA Next, the ‘push’ model empowers the device managers, software components, etc., to push registration information to the managing element. This reduces the communication required between components and even permits ‘static’ configurations that are precompiled and permitting ‘nearly-instant’ startups.

As a change from SCA 2.2.2, a CORBA naming service is no longer required and not used by the core framework or other SCA-compliant components. Instead, the Application Factory as shown in Figure 7 maintains a component registry of the components.

When a waveform is to be instantiated, the application factory reads a sequence of XML files (which could be pre-parsed) to determine what software components need to be launched and where the components are to be distributed. As illustrated in Figure 7, the Application Factory will send load and execute commands to the processor (SCA Device) that will be hosting the waveform component. Once the waveform component has been instantiated, it will register itself with the component registry so that the application factory can configure and connect that component with other software components.

The Assembly Controller (AC) in Figure 7 is a waveform component defined by the SCA to receive commands from the domain manager and radio set command and control. This focuses all waveform control to a single component, which can delegate commands as required.

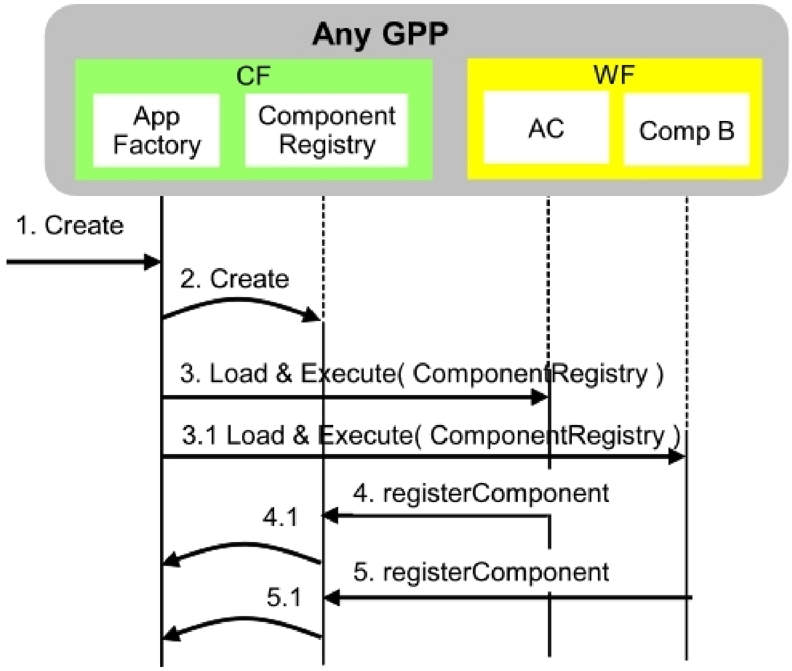

Figure 7: Registration for the SCA Full Profile

Figure 7 illustrates registration is a 'push' model rather than 'pull', which is the new SCA design pattern. Previous versions required the application factory and Device Managers to 'pull' registration from the software elements of the radio. In this 'push' model, components are responsible for reporting directly to the registry. This reduces communication traffic and also enables a least-privilege architecture – components have the minimum amount of visibility or control to execute their functionality.

2.4.4 Application Factory

In Figure 7, the Application Factory creates the software components by sending instructions to the Devices representing the processors. After the components have been instantiated, the Application Factory will send 'connect' commands to the components, providing them the object references necessary for them to communicate with the desired component. The Application Factory reads the Software Assembly Descriptor (SAD) file, which is the logical equivalent of a schematic diagram to 'wire' the waveform together.

SCA 2.2.2 user experiences with port connections resulted in changes for SCA Next. The new version of the SCA allows the legacy-type connections, shown as *Connection Sequence for Obtainable Ports* in Figure 8. With this design model, the Application Factory queries every component for its connection IDs to provides ports so the application factory can send them to components that require connection. A slight modification will be required to legacy waveforms to incorporate the revised port architecture of SCA Next.

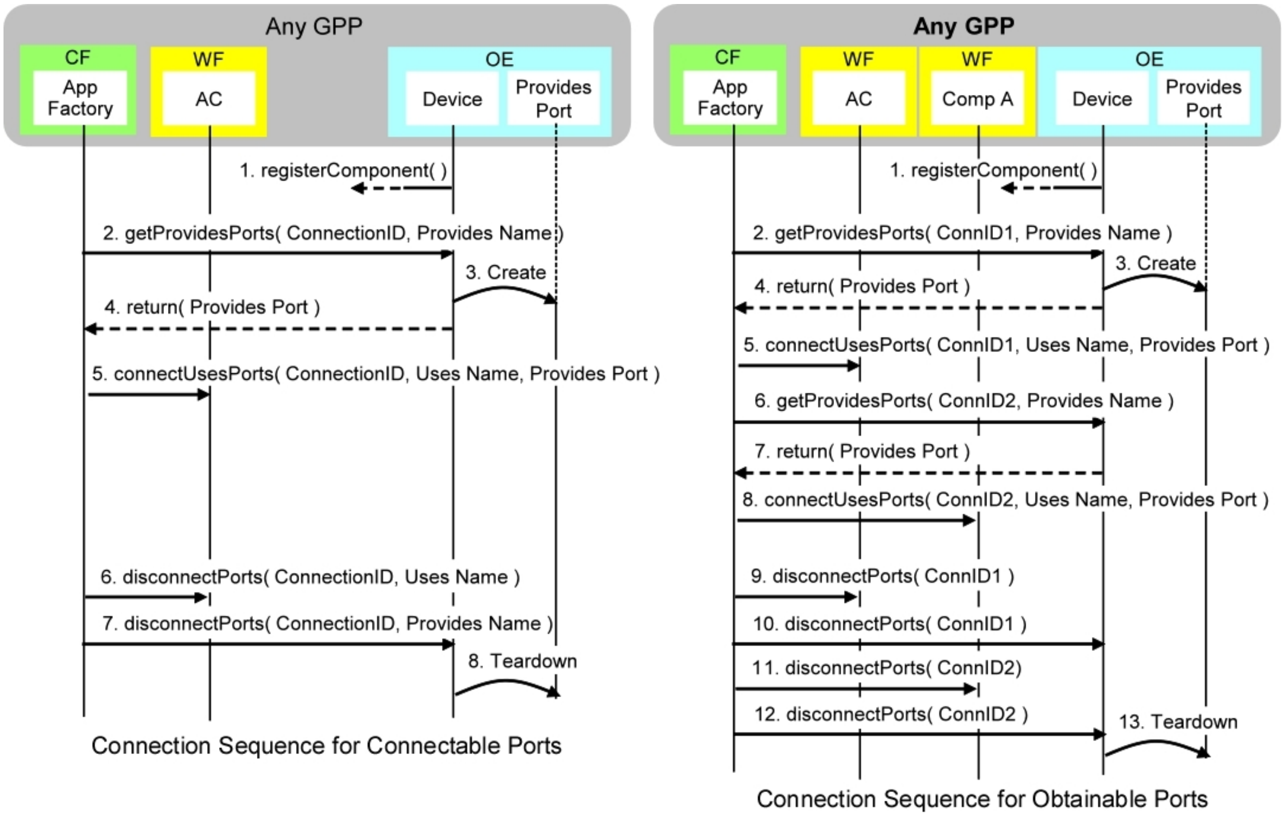

Figure 8: Different Types of Port Connections

The connectable port option for SCA permits components to return their connection IDs upon registration, eliminating the extra communication traffic required for the *getProvidesPorts* operation. This new method is not as flexible, or plug-and-play, but it improves startup times for waveforms.

A third variation is where the application factory reads an XML file containing precompiled connection IDs generated at build time. The application factory does not require registration from these components. Thus when the components come to life, they are already pre-wired and ready for waveform operation.

2.4.5 Lightweight Components

Users commented that SCA 2.2.2 interfaces were a one-size-fits-all, resulting in some components being larger than necessary. For example, a SCA 2.2.2 resource component includes testable objects, properties, etc. But what if a component doesn't need a self-test or properties? SCA 2.2.2 still required the component to implement that functionality.

SCA Next introduces a new design pattern – optional inheritance. The new Resource interface is illustrated in Figure 9. Only one interface is mandatory – Life Cycle. Other interfaces are available as necessary. Precompiler definitions in the IDL permit developers to specify which interfaces a specific component requires. Because of this feature, SCA Next components should be smaller than previous versions.

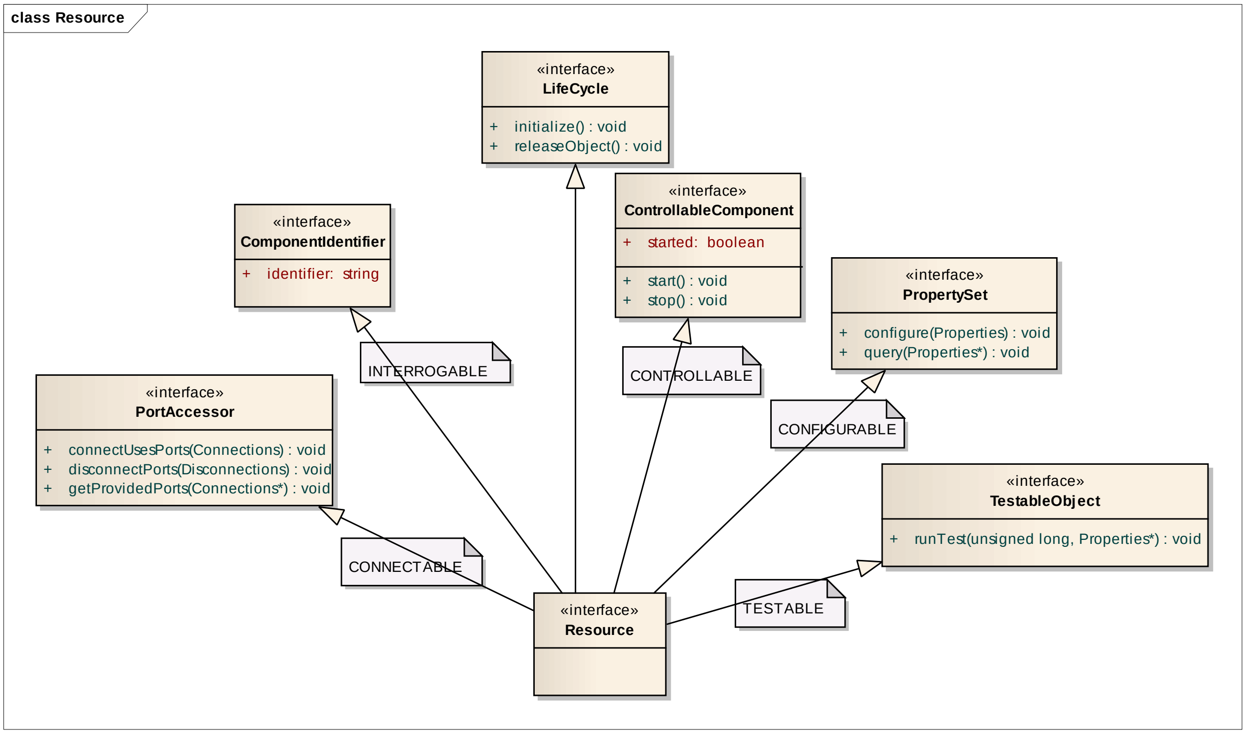

Figure 9: Resource Interface Features Optional Inheritance

SCA Next defines a legacy flag which permits developers to turn on all the interfaces as with previous versions of the SCA. As mentioned earlier, SCA Next has reworked the port interfaces. Earlier versions had two port interfaces – port and portSupplier. SCA Next has deleted those interfaces in favor of a single PortAccessor interface which combines the functionality. Legacy SCA 2.2.2 waveforms will require updating to the new PortAccessor interface, but other modifications should not be necessary.

The optional inheritance design pattern has been extended elsewhere within other core framework interfaces. For example, the LoadableCapacity interface was previously required of all loadableDevices. In some implementations, the functionality provided by this interface may be unused, so SCA Next permits the component to be constructed without it.

2.5 EXAMPLE SCA WAVEFORM

The publicly available FM3TR waveform architecture is illustrated in Figure 10. (This waveform is available from the JTRS Open Source Information Repository [4].) The yellow-colored components represent radio set functionality, whereas the red and blue colored blocks represent waveform software components. The SCA defines port connections, or interfaces for each macro-sized component. (The Data Link Control (DLC) component could be an aggregation of many smaller pieces, but only exposes the minimal interface connections to the at-large system.)

The blue and red software components on the GPP expose: in, out, and control ports. These are interfaces that can be connected to other software components, radio services, or radio set hardware devices. Generally, the ‘in’ ports are described as ‘provides’ ports, whereas the ‘out’ ports are ‘uses’ ports, because they either provide or use port connections, respectively.

Using either the middleware services provided by the radio set, or direct C++ pointers, connection IDs and object references permit independent software components to communicate. The components only need each other’s pointer or object reference. The messaging becomes more difficult if the components are distributed into separate memory partitions. For such deployments, middleware services allow a general solution to be applied throughout the complete radio set.

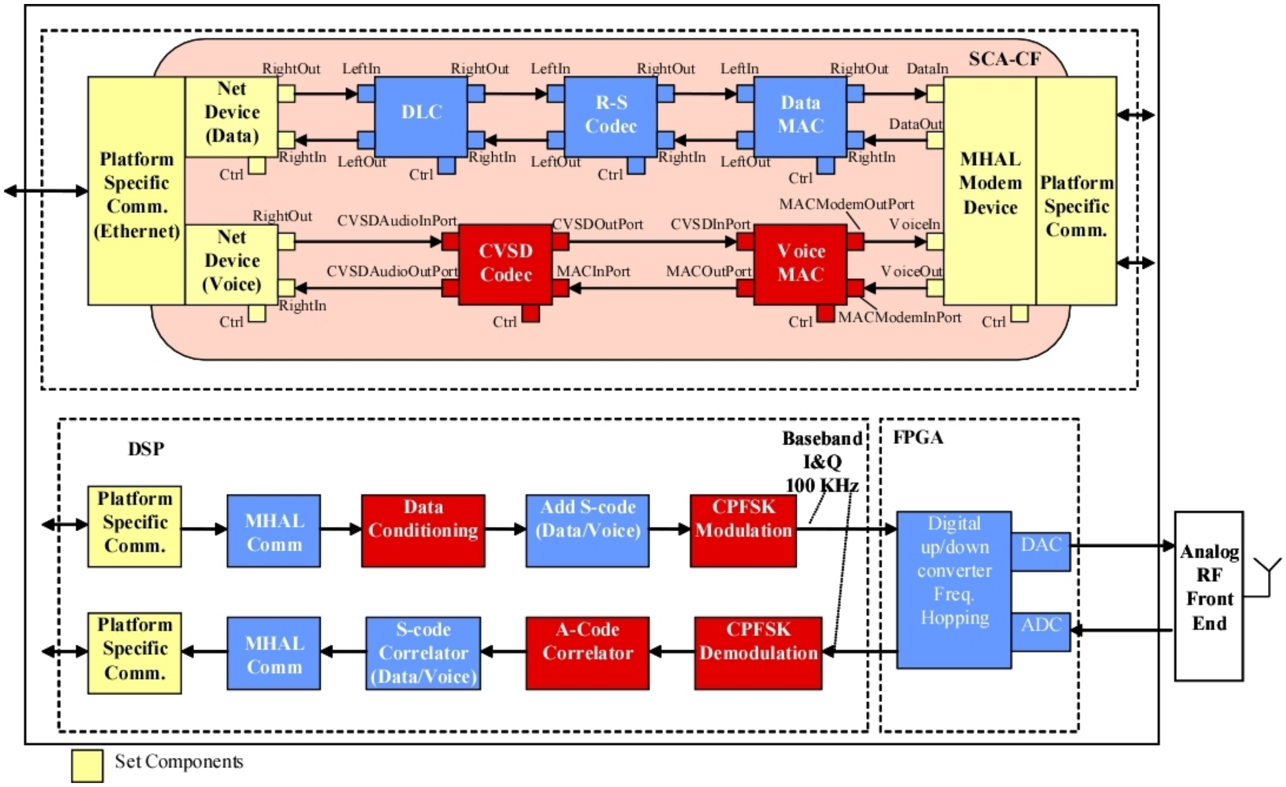

Figure 10: Example SCA Waveform

The FM3TR waveform is a simple time domain multiplexed access (TDMA) application with Continuous Phase Frequency Shift Keying (CPFSK) as the baseband modulation. The JTRS implementation provides either data or voice operation. Continuously Variable-Slope Delta modulation (CVSD) is implemented for the vocoder. Reed-Solomon (R-S) forward error coding is used to improve the bit reliability of the wireless link.

The Data Multiple Access Control (MAC) is an SCA resource that converts the input data stream into data symbols grouped to match the R-S coding format. The voice MAC performs a similar operation for the data stream produced by the vocoder. The A-code is a simple 32-bit synchronization code used to synchronize transmitter and receiver. The S-code is a second synchronization word used to identify data packet types such as voice, data, etc.

The architecture and deployment of this waveform is fairly typical for SCA implementations, although other variations are possible. In this example, the waveform components within the FPGA and DSP do not have SCA interfaces. Historically radio architects have attempted to wring the last drop of performance from the DSP and FPGA devices and not implemented SCA interfaces on these lower-level software components. There is a substantial cost for this strategy – a loss of portability for these waveform components.

SCA Next introduces the lwAEP which restores portability for these software components without unnecessarily burdening the DSP and FPGA processing elements. The MHAL On Chip Bus (MOCB) specification allows better utilization of DSP and FPGA resources than the MHAL API, replacing the *pushpacket* operation model with a shared memory ‘pull’ architecture. MOCB is recommended for new designs.

An example deployment of FM3TR is illustrated in Figure 11 with the radio devices, radio services, and core framework components.

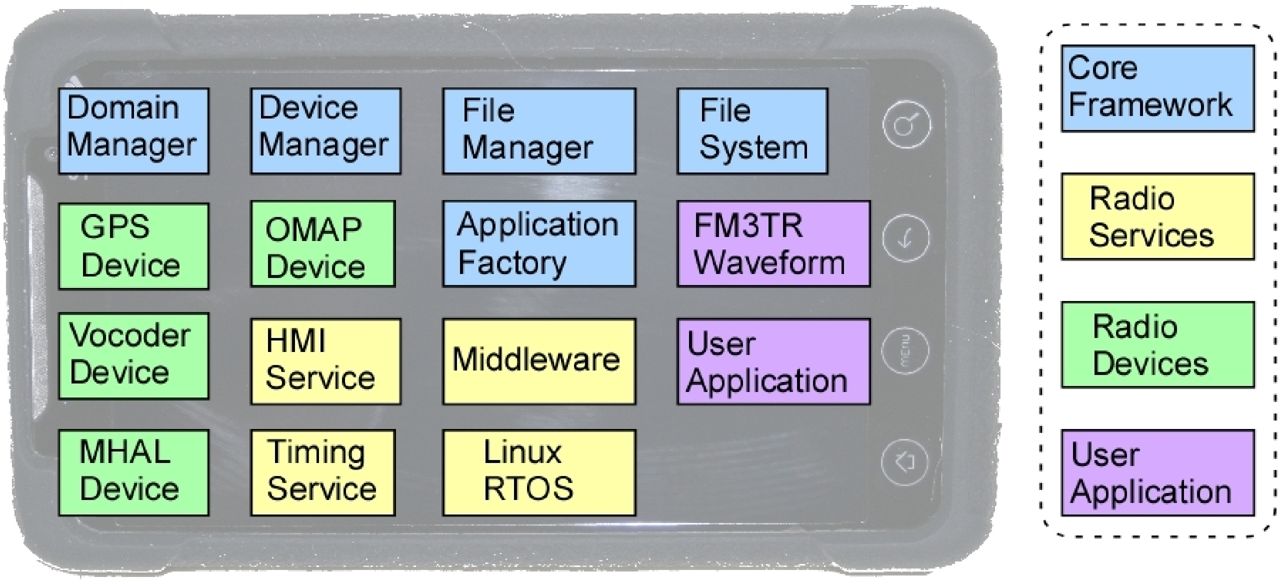

Figure 11: Example Deployment of FM3TR

3 SCA NEXT RESPONSIBILITIES

In the body of the specification, the figure - Core Framework IDL Relationships accurately depicts complete set of SCA Next interfaces and their inheritances. Because of its history, the SCA Next documentation blurs discussion between the components hosted by the radio set and those provided by waveforms. Figure 12 attempts to identify specific interfaces of interest to the various stakeholders in a radio set architecture.

The base SCA interfaces will probably be used throughout the radio, even with device and framework implementations. But the absence of ‘shalls’ requiring these interfaces disqualify an ‘x’ for the interface.

	Base Interfaces						Device Interfaces											Framework Services												OE Services										
	PortAccessor	LifeCycle	TestableObject	PropertySet	Resource	ComponentIdentifier	ComponentManager	ComponentFactory	ControllableComponent	Device	LoadableDevice	ExecutableDevice	AggregateDevice	LoadableObject	DeviceAttributes	CapacityManager	ManageableComponent	Application	ApplicationFactory	ApplicationDeploymentData	DomainManager	DomainInstallation	ComponentRegistry	FullComponentRegistry	ManagerRegistry	FullManagerRegistry	DeviceManager	ManagerRelease	DeviceManagerAttributes	FileManager	FileSystem	File	Event Service	Log Service	User Defined Service	Domain Profile	Operating System	Middleware Services	Applications	Logical Devices
Application Developers	X	X	X	X	X	X	X	X	X																											X			X	
Core Developers																		X	X	X	X	X	X	X	X	X	X	X	X							X				
Device Developers	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X																			X				X
Service Developers																														X	X	X	X	X	X					
System Developers																																	X	X	X		X	X		

Figure 12: General Allocation of Interfaces to Radio Developers

Another helpful perspective of SCA Next is the component hierarch diagram illustrated in Figure 13.

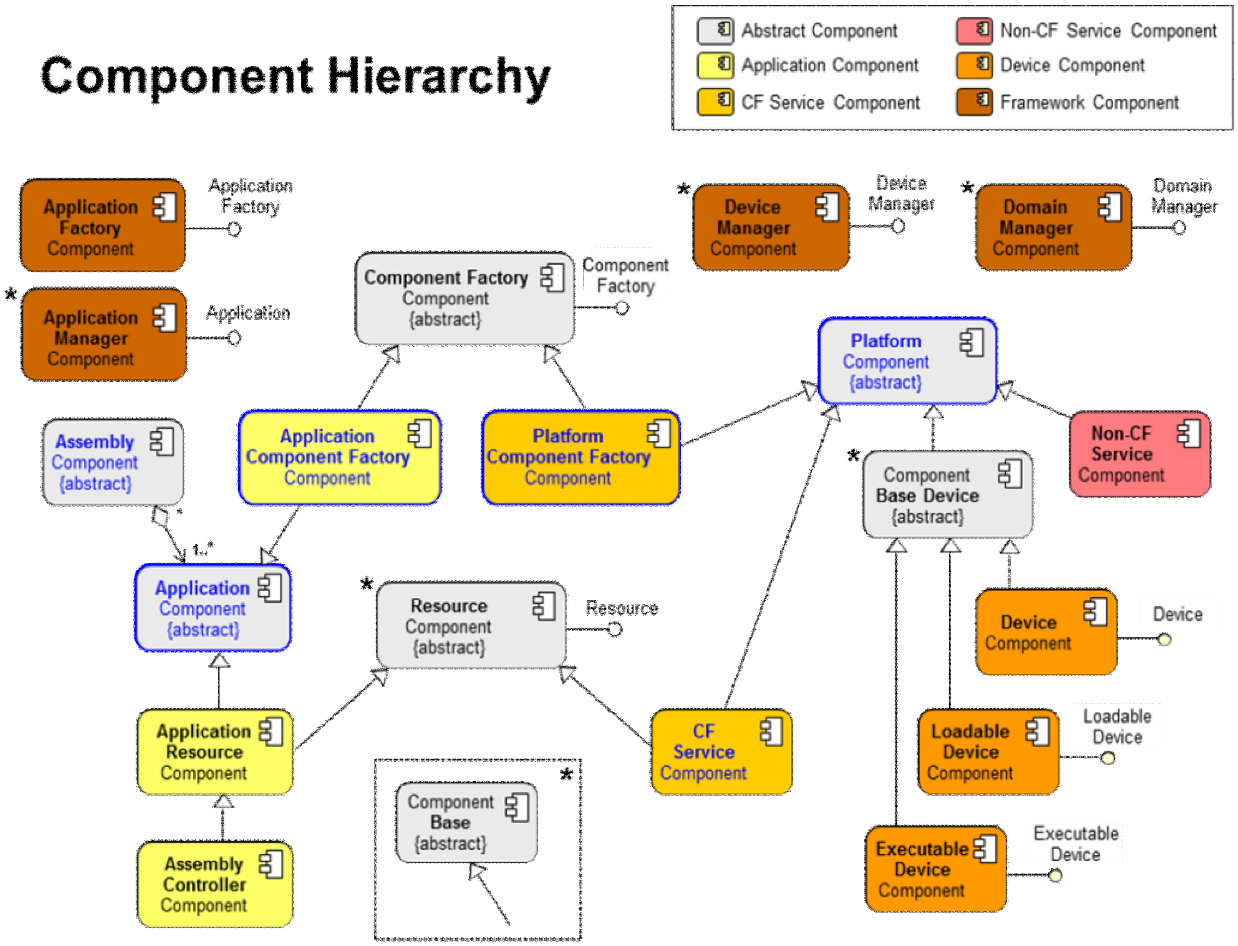

Figure 13: SCA Next Component Hierarchy

4 REFERENCES

1 JTRS SCA Website, http://sca.jpeojtrs.mil/home.asp

2 Donald R. Stephens, Cinly Magsombol, Chalena Jimenez, "Design patterns of the JTRS infrastructure", MILCOM 2007 - IEEE Military Communications Conference, no. 1, October 2007, pp. 835-839

3 Cinly Magsombol, Chalena Jimenez, Donald R. Stephens, "Joint tactical radio system—Application programming interfaces", MILCOM 2007 - IEEE Military Communications Conference, no. 1, October 2007, pp. 855-861

4 JTRS Open Source Information Repository, http://gforge.calit2.net/gf/project/jtrs_open_ir/

5 Donald R. Stephens, Rich Anderson, Chalena Jimenez, Lane Anderson, "Joint tactical radio system—Waveform porting", MILCOM 2008 - IEEE Military Communications Conference, vol. 27, no. 1, November 2008, pp. 2629-2635

6 JTRS Waveform Portability Guidelines, http://sca.jpeojtrs.mil/portabilityguidelines.asp

7 Donald R. Stephens, Cinly Magsombol, Norman Browne, "Network programming of joint tactical radio system radios", MILCOM 2008 - IEEE Military Communications Conference, vol. 27, no. 1, November 2008, pp. 2623-2628

5 ACRONYMS

Abbreviation	Definition
AC	Assembly Controller
AEP	Application Environment Profile
API	Application Program Interface
CF	Core Framework
CORBA	Common Object Request Broker Architecture
CPFSK	Continuous Phase Frequency Shift Keying
CVSD	Continuously Variable-Slope Delta modulation
DLC	Data Link Control
DSP	Digital Signal Processor
FPGA	Field Programmable Gate Array
GPP	General Purpose Processor
HCI	Human-Computer Interface
ID	Identification, Identifier
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronic Engineers
IOR	Interoperable Object Reference
ISO	International Standards Organization
JPA	JTRS Platform Adapter
lwAEP	Lightweight Application Environment Profile
MHAL	Modem Hardware Abstraction Layer
MOCB	MHAL On Chip Bus
OE	Operating Environment
OMG	Object Management Group
ORB	Object Request Broker
OS	Operating System

Abbreviation	Definition
OMAP	Open Multimedia Application Platform
POSIX®	Portable Operating System Interface
RPC	Remote Procedure Control
R-S	Reed Solomon
SAD	Software Assembly Descriptor
SCA	Software Communications Architecture
SW	Software
TCP-IP	Transmission Control Protocol (TCP) and Internet Protocol (IP)
TDMA	Time Division Multiplexed Access
UML	Unified Modeling Language
XML	eXtensible Markup Language

® POSIX is a registered trademark of the Institute of Electrical and Electronics Engineers, Inc.